Parallel PageRank Computation using MPI

CSE 633 Parallel Algorithms (Fall 2012)
Xiaoyi (Eric) Li
Email: xiaoyili@buffalo.edu
Outline

- Markov Chains
- PageRank Computation
- Parallel Algorithm
- Message Passing Analysis
- Experiments and result
Markov Chains

- **Markov Chain:**
  - A Markov chain is a discrete-time stochastic process consisting of N states.

- **Transition Probability Matrix:**
  - A Markov chain is characterized by an N*N transition probability matrix P.
  - Each entry is in the interval [0,1].
  - A matrix with non-negative entries that satisfies $\forall i, \sum_{j=1}^{N} P_{ij} = 1$.
  - Chain is acyclic
  - There is a unique steady-state probability vector $\pi$.
    - $\eta(i,t)$ is the number of visits to state $i$ in $t$ steps.
    - $\pi(i)>0$ is the steady-state probability for state $i$. 
      $ \lim_{t \to \infty} \frac{\eta(i,t)}{t} = \pi(i) $
PageRank Computation

- **Target**
  - Solve the steady-state probability vector $\pi$, which is the PageRank of the corresponding Web page.

- **Method**
  - Iteration.
  - Given an initial probability distribution vector $x_0$
    - $x_0 \cdot P = x_1$, $x_1 \cdot P = x_2$ … Until the probability distribution converges. (Variation in the computed values are below some predetermined threshold.)
Practical PageRank Calculation

\[
PR(p_i) = \frac{1 - d}{N} + d \sum_{p_j \in M(p_i)} \frac{PR(p_j)}{L(p_j)}
\]
Parallel Algorithm
**Initialization**

- **Master**
  - Received individual index, initialize send & receive buf for each worker.
  - Initialize global weights, send weights\[\text{index}_i\] to workers\_i

---

**Begin iteration**

- **Master**
  - Gather individual updates from workers, update the global weight determined by \text{index}_i
  - Check convergence
  - If not, send global weights to workers
  - If yes, send stop signal and do housekeeping

---

**Worker**

- Read bucket, construct local graph and send two index -- node to update & node required to master

- Update local graph using received weight. Calculate PageRank once.
- Send the updated score back to master.

---
Total number of iterations
Message Passing Analysis

**Without weight index**

- Each worker send & receive global weight from master:
  - 1M web-nodes, 64 workers:
    - 2 * 8 bytes * 1M = 16MB
    - 16 * 64 = 1024MB = 1GB
    - Total = #iteration * 1GB

**With weight index**

- Each worker send & receive global weight from master:
  - 1M web-nodes, 64 workers:
    - Send: 8 * 1M / #workers ≈ 0.128MB
    - Rec: 8 * 1M / (small fraction, e.g #nodes/8) ≈ 1MB
    - 1.128 * 64 ≈ 72MB
    - Total = #iteration * 72MB
Experiments

- Data: wiki-votes (67035 | 1025563)
- \#nodes = 32, IB2, ppn=4
## Results

<table>
<thead>
<tr>
<th>#cores</th>
<th>run time (ms)</th>
<th>speed up</th>
<th>efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>519</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>269</td>
<td>1.92936803</td>
<td>0.96468401</td>
</tr>
<tr>
<td>4</td>
<td>173</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>148</td>
<td>3.50675676</td>
<td>0.87668919</td>
</tr>
<tr>
<td>6</td>
<td>123</td>
<td>4.2195122</td>
<td>0.84390244</td>
</tr>
<tr>
<td>7</td>
<td>96</td>
<td>5.40625</td>
<td>0.90104167</td>
</tr>
<tr>
<td>8</td>
<td>87</td>
<td>5.96551724</td>
<td>0.85221675</td>
</tr>
<tr>
<td>16</td>
<td>45</td>
<td>11.53333333</td>
<td>0.76888889</td>
</tr>
<tr>
<td>32</td>
<td>25</td>
<td>20.76</td>
<td>0.66967742</td>
</tr>
<tr>
<td>64</td>
<td>38</td>
<td>13.6578947</td>
<td>0.21679198</td>
</tr>
<tr>
<td>128</td>
<td>74</td>
<td>7.01351351</td>
<td>0.05522452</td>
</tr>
</tbody>
</table>
Results

Run time

Running time (ms)

Number of cores
Results

Speedup

Number of cores

Speedup

0 20 40 60 80 100 120 140

0 5 10 15 20 25

Results

Efficiency

Number of cores

Efficiency

0  20  40  60  80  100  120  140

0  0.2  0.4  0.6  0.8  1  1.2

Efficiency decreases as the number of cores increases.
Questions?